How SPF Works

In a link-state routing protocol, each router knows about all other routers in a network and the links that connect these routers. In OSPF, this information is encoded as Link-State Advertisements (LSAs); in IS-IS, this information is Link-State Packets (LSPs). This chapter does not discuss the differences between these two protocols. See Appendix B,"CCO and Other Resources," for more information. Because the acronym LSP also stands for Label-Switched Path, the unit of information a router uses to flood its connectivity information will be called an LSA. Remember that IS-IS works much the same way that OSPF does.

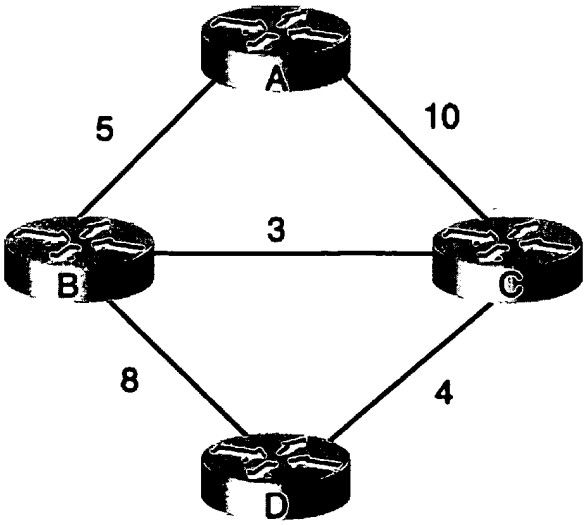
As soon as a router knows about all other routers and links, it runs the *Dijkstra Shortest Path First* algorithm (named after Edsger W. Dijkstra, who first identified it) to determine the shortest path between the calculating router and all other routers in the network.

Because all routers run the same calculation on the same data, every router has the same picture of the network, and packets are routed consistently at every hop.

But how does the Dijkstra calculation actually work? It's actually quite simple. Understanding SPF is fundamental to understanding MPLS Traffic Engineering's CSPF, which is based on the basic Dijkstra SPF algorithm.

Suppose you have a simple network similar to the one shown in Figure 4-1.

Figure 4-1. Simple Network Topology Demonstrating the SPF Algorithm



This example walks through what happens when Router A runs SPF and generates its routing table.

After each router has flooded its information to the network, all the routers know about all the other routers and the links between them. So the link-state database on every router looks like Table 4-1.

Table 4-1. Class Map Matches

Router	{neighbor, cost} Pairs
A	{B,5} {C,10}
B	{A,5} {C,3} {D,8}
C	{A,10} {B,3} {D,4}
D	{B,8} {C,4}

So what does Router A do with this information?

In SPF calculation, each router maintains two lists:

- A list of nodes that are known to be on the shortest path to a destination. This list is called the PATH list or sometimes the PATHS list. It is important to understand that the only things on the PATH list are paths to a destination that are known to be the shortest path to that destination.

- A list of next hops that might or might not be on the shortest path to a destination. This list is called the TENTatitve or TENT list.

Each list is a table of {router, distance, next-hop} triplets from the perspective of the calculating router.

Why a triplet? You need to know which router you're trying to get to in PATH and TENT lists, so the first part of the triplet is the name of the router in question. In reality, this is the router ID (RID) rather than the textual name, but for this example, we'll use the router name. The next hop is the router you go through to get to the node in question. The third item in the triplet is the distance, which is the cost to get to the node in question.

Why are the names of these lists in all capital letters? That's just how they're written. They're not acronyms; they're just written in capitals. Most other examples of SPF calculation, and many debugs involving the SPF calculations themselves, refer to these lists as PATH (or sometimes PATHS) and TENT.

The algorithm for computing the shortest path to each node is simple. Each router runs the following algorithm:

Step 1. Put "self" on the PATH list with a distance of 0 and a next hop of self. The router running the SPF refers to itself as either "self" or the *root node,* because this node is the root of the shortest-path tree.

Step 2. Take the node just placed on the PATH list, and call it the PATH node. Look at the PATH node's list of neighbors. Add each neighbor in this list to the TENT list with a next hop of the PATH node, unless the neighbor is already in the TENT or PATH list with a lower cost. Call the node just added to the TENT list the TENT node. Set the cost to reach the TENT node equal to the cost to get from the root node to the PATH node plus the cost to get from the PATH node to the TENT node. If the node just added to a TENT list already exists in the TENT list, but with a higher cost, replace the higher-cost node with the node currently under consideration.

Step 3. Find the neighbor in the TENT list with the lowest cost, add that neighbor to the PATH list, and repeat Step 2. If the TENT list is empty, stop.

This might seem complicated, but it's really straightforward. Consider the process that Router A in Figure 4-1 goes through to build its routing table:

Step 1. Put "self" on the PATH list with a distance of 0 and a next hop of self.

This means that Router A's databases look like Table 4-2.

Table 4-2. PATH and TENT Lists for Router A

PATH List	TENT List
{A,0,0}	(empty)

Step 2. Take the node just placed on the PATH list, and call it the PATH node. Look at that node's neighbor list. Add each neighbor in this list to the TENT list with a next hop of the PATH node, unless the neighbor is already in the TENT or PATH list with a lower cost. If the node just added to a TENT list already exists in the list, but with a higher cost, replace the higher-cost node with the node currently under consideration.

In this example, {B,5,B} and {C,10,C} get added to the TENT list, as reflected in Table 4-3.

Table 4-3. PATH and TENT Lists for Router A After Step 2

PATH List	TENT List
{A,0,0}	{B,5,B}
	{C,10,C}

Step 3. Find the neighbor in the TENT list with the lowest cost, add that neighbor to the PATH list, and repeat Step 2. If the TENT list is empty, stop.

{B,5,B} is moved to the PATH list, because that's the shortest path to B. Because {C,10,C} is the only other neighbor of Router A, and the cost to get to C is greater than the cost to get to B, it's impossible to ever have a path to B that has a lower cost than what's already known. Table 4-4 reflects the PATH and TENT lists at this point.

Table 4-4. PATH and TENT Lists for Router A After Step 3

PATH List	TENT List
{A,0,0}	{C,10,C}
{B,5,B}	

Step 4. Repeat Step 2. Take the node just placed on the PATH list and call it the PATH node. Look at that node's neighbor list. Add each neighbor in this list to the TENT list with a next hop of the PATH node, unless the neighbor is already in the TENT or PATH list with a lower cost. If the node just added to a TENT list already exists in the list, but with a higher cost, replace the higher-cost node with the node currently under consideration.

Router B's neighbors are examined. Router B has a link to C with a cost of 3 and a link to D with a cost of 8. Router C, with a cost of 5 (to get from "self" to B) + 3 (to get from B to C) =

8 (the total cost from A to C via B) and a next hop of B, is added to the TENT list, as is Router D, with a cost of 5 (the cost to get from the root node to B) + 8 (the cost to get from B to D) = 13 and a next hop of B. Because the path to C with a cost of 8 through B is lower than the path to C with a cost of 10 through C, the path to C with a cost of 10 is removed from the TENT list. Table 4-5 reflects the PATH and TENT lists at this point.

Table 4-5. PATH and TENT Lists for Router A After Step 4

PATH List	TENT List
{A,0,0}	~~{C,10,C}~~
{B,5,B}	{C,8,B}
	{D,13,B}

Step 5. Find the path in the TENT list with the lowest cost, add that path to the PATH list, and repeat Step 2. If the TENT list is empty, stop.

The path to C through {C,8,B} is moved from the TENT list to the PATH list, as reflected in Table 4-6.

Table 4-6. PATH and TENT Lists for Router A After Step 5

PATH List	TENT List
{A,0,0}	{D,13,B}
{B,5,B}	
{C,8,B}	

Step 6. Take the path just placed on the PATH list, and look at that node's neighbor list. For each neighbor in this list, add the path to that neighbor to the TENT list, unless the neighbor is already in the TENT or PATH list with a lower cost. If the node just added to a TENT list already exists in the list, but with a higher cost, replace the higher-cost path with the path currently under consideration.

Under this rule, the path to D through B (which is really B⟶C⟶D) with a cost of 12 replaces the path to D through B⟶D with a cost of 13, as reflected in Table 4-7.

Table 4-7. PATH and TENT Lists for Router A After Step 6

PATH List	TENT List
{A,0,0}	~~{D,13,B}~~
{B,5,B}	{D,12,B}
{C,8,B}	

Step 7. Find the neighbor in the TENT list with the lowest cost, add that neighbor to the PATH list, and repeat Step 2. If the TENT list is empty, stop.

The path to D is moved to the PATH list, as reflected in Table 4-8.

Table 4-8. PATH and TENT Lists for Router A After Step 7: The TENT List Is Empty

PATH List	TENT List
{A,0,0}	
{B,5,B}	
{C,8,B}	
{D,12,B}	

Step 8. Find the neighbor in the TENT list with the lowest cost, add that neighbor to the PATH list, and repeat Step 2. If the TENT list is empty, stop.

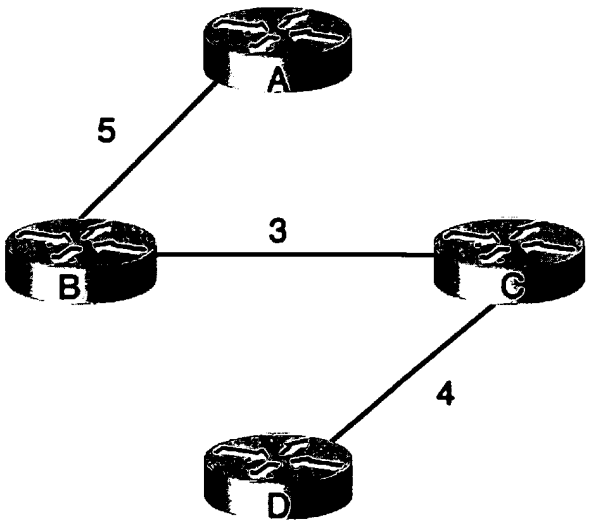
The TENT list is empty because D has no neighbors that are not already on the PATH list, so you stop. At this point, the PATH list becomes A's routing table, which looks like Table 4-9.

Table 4-9. Router A's Routing Table

Node	Cost	Next Hop
A	0	Self
B	5	B (directly connected)
C	8	B
D	12	B

So that's how the basic SPF algorithm works. After it's done, the topology (according to Router A's routing table) looks like Figure 4-2.

Figure 4-2. Router A's View of the Network After Running the SPF Algorithm



As you can see, traffic from Router A never crosses the links from A to C or the links from B to D.

Two things should jump out at you. The first is that the only traffic that crosses the link from B to D is traffic from Router A.

The second is that the link from Router A to Router C is not used at all, because its cost (10) is so high. In a real network, a link like this would end up being expensive dead weight, only being used when another link in the network fails. The actual SPF implementation is a bit more complex than what's been described here, but this is the general idea.

PREVIOUS < Free Open Study > NEXT